

# **Was bringt der „Merced“?**

---

## **Die Prozessorarchitektur „IA-64“**

**Bernd Däne**

TU Ilmenau, Fakultät I/A

Tel.: 03677-69-1433

[bdaene@theoinf.tu-ilmenau.de](mailto:bdaene@theoinf.tu-ilmenau.de)

# Gliederung

---

1. Merced - Itanium - IA-64
2. Die Vorgeschichte
3. Der neue Ansatz
4. Die Details
5. Die Zukunft
6. Die Anderen
7. Literatur und Web

Hinweis:  
Eingetragene Warenzeichen sind Eigentum der jeweiligen Firmen.  
Die Programmbeispiele beruhen auf Veröffentlichungen der Intel Corp.

# 1. Merced - Itanium - IA-64

---

## ■ „Merced“:

- ‚Deckname‘ für Intel's 64-bit-Projekt
- Schon lange bekannt, aber wenig Infos

## ■ „IA-64“:

- Bezeichnung des Programmiermodells
- Seit Mai 1999 vollständig veröffentlicht

## ■ „Itanium“:

- ‚Offizieller‘ Name des ersten Prozessors
- Partielle Vorstellung im Oktober 1999

# 2. Die Vorgeschichte

---

## ■ Desktop-Systeme:

- Intel-Familie: **PC** (großer Marktanteil)
- Motorola-Familie: **Mac** (kleinerer Marktanteil)

## ■ Server und High-End-Workstations:

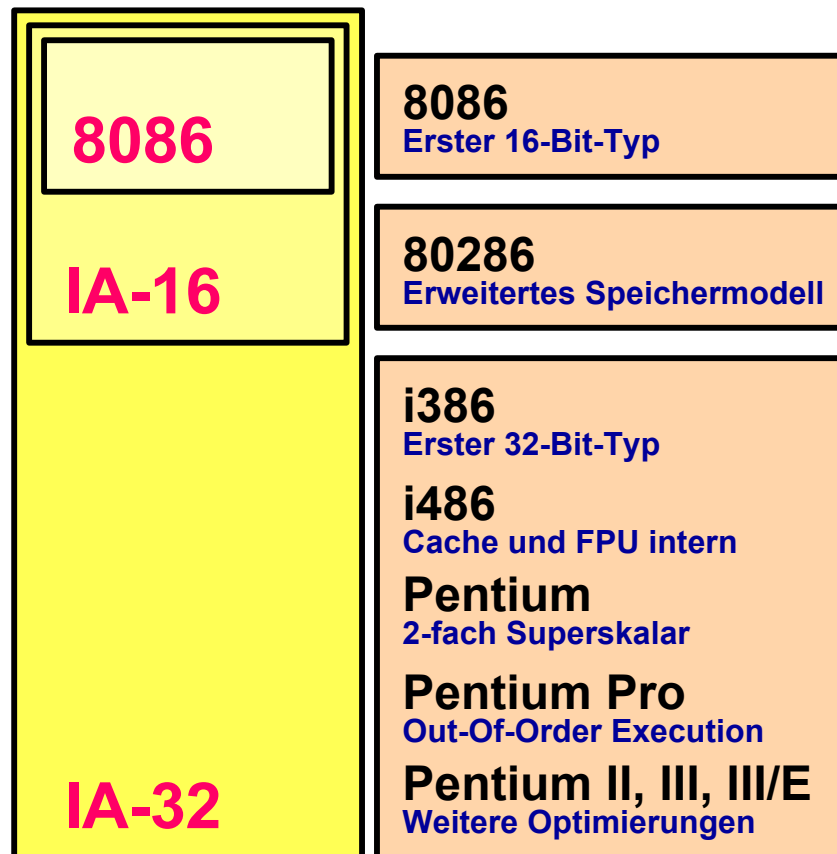
- Alternative Architekturen (Nische, abnehmend)
- Desktop-Familien (zunehmend)

## ■ Eingebettete Systeme:

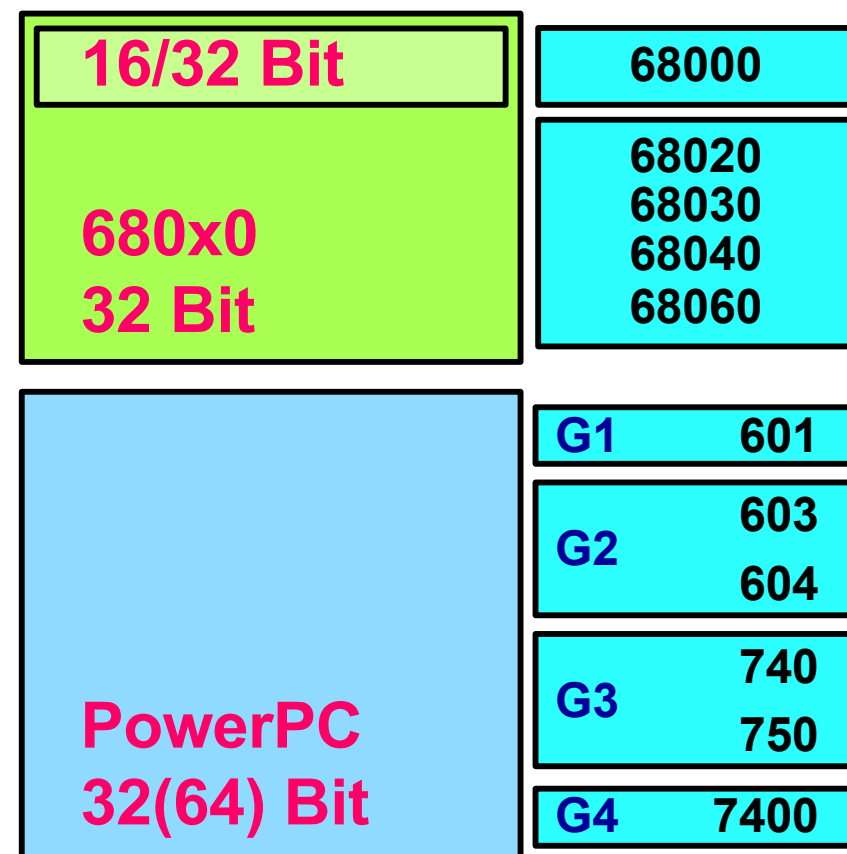
- Große Vielfalt (Mikrocontroller, Digitale Signalprozessoren, Ableittypen aus Desktop-Familien)

# Intel-Familie und Motorola-Familie

## Intel



## Motorola



# Nachteile der Architektur IA-32

---

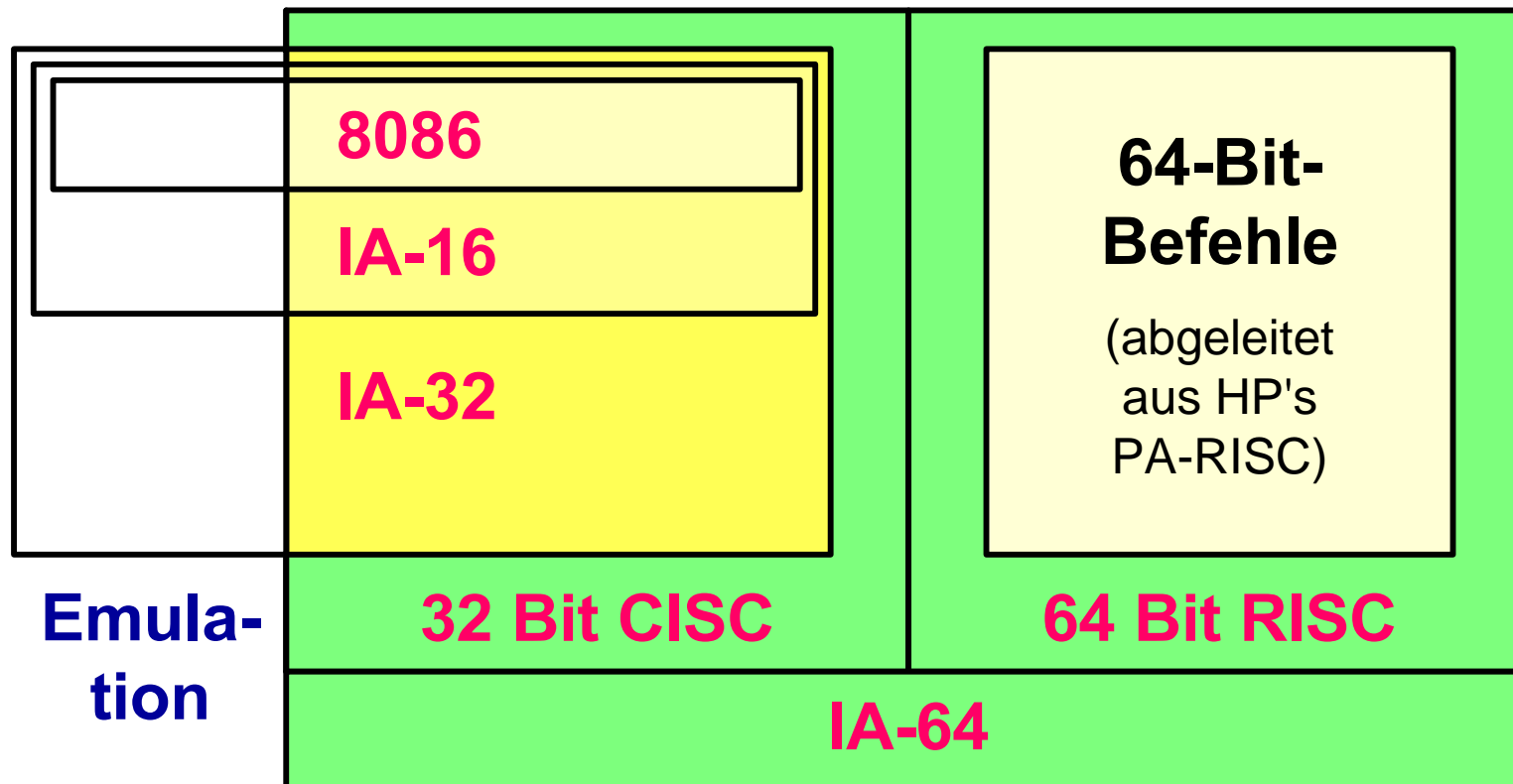
- **Komplexes Programmiermodell**
  - Befehle mit sehr unterschiedlichem Funktionsumfang
  - Viele Ausnahmen und Irregularitäten
- **Komplizierte Befehlscodes**
  - Unterschiedliche Länge (1 bis 14 Bytes)
  - Viele verschiedene Formate
- **Sehr wenig Register**
- **Komplizierte Speicherverwaltung**



**Schwierige Optimierung von Mikroarchitektur und Compilern**

# 3. Der neue Ansatz

---



# Kompatibilität bei IA-64

---

- **64-Bit-Applikation und 64-Bit-OS:**
  - **o.k.**
- **16/32-Bit-Applikation und 64-Bit-OS:**
  - **binärkompatibel**, falls vom OS unterstützt
- **16/32-Bit-Applikation und 16/32-Bit-OS:**
  - **binärkompatibel** mit Zusatzsoftware



# CISC und RISC

---

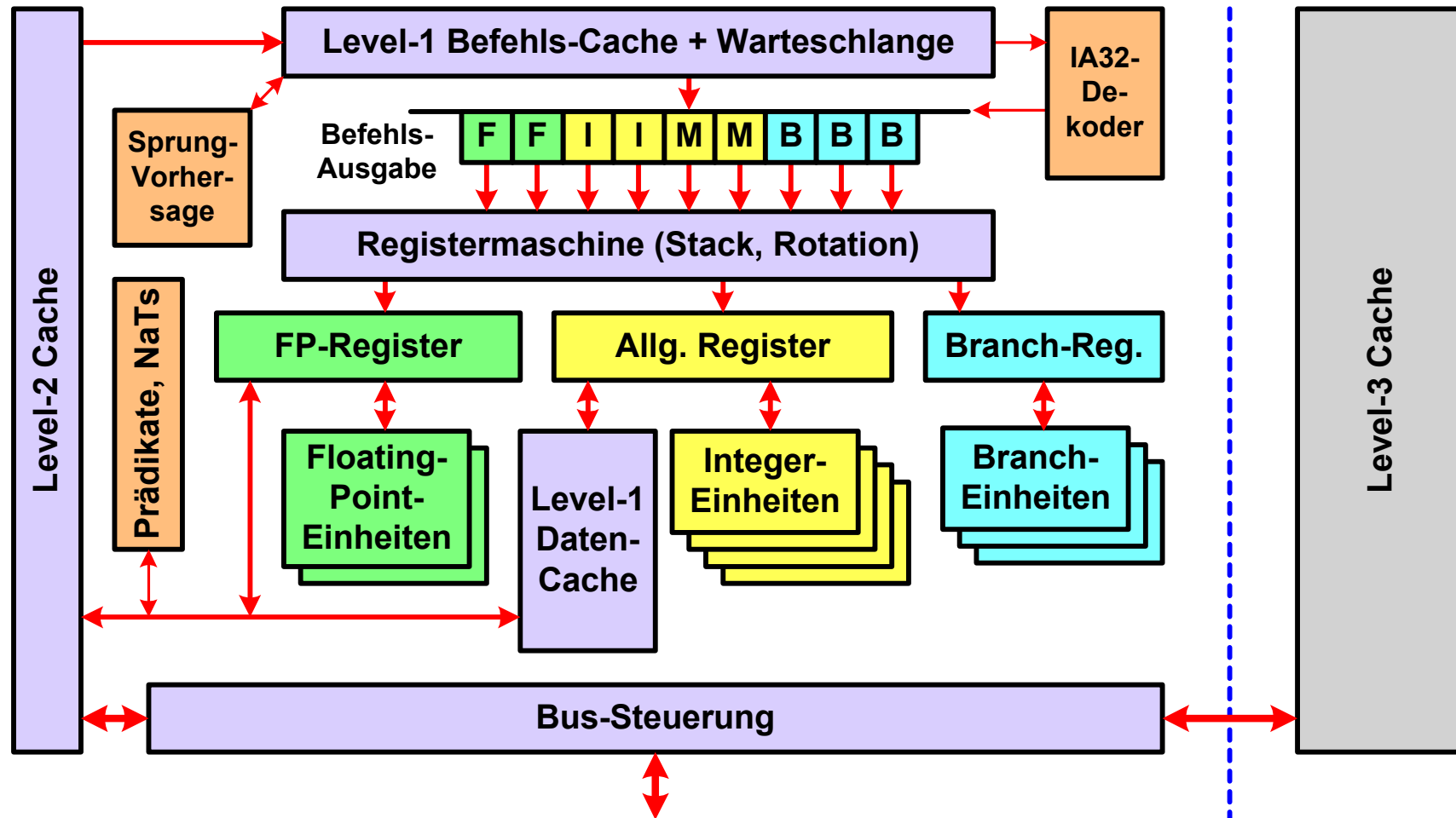
<b>CISC</b>	<b>RISC</b>
<b>Complex Instruction Set Computing</b>	<b>Reduced Instruction Set Computing</b>
<b>Einfache und komplexe Befehle</b>	<b>Nur einfache Befehle</b>
<b>Heterogener Befehlssatz</b>	<b>Orthogonaler Befehlssatz</b>
<b>Verschiedene Taktzahl pro Befehl</b>	<b>Meist 1 Takt pro Befehl</b>
<b>Viele Befehlscode-Formate mit unterschiedlicher Länge</b>	<b>Wenige Befehlscode-Formate mit einheitlicher Länge</b>

# 4. Die Details

---

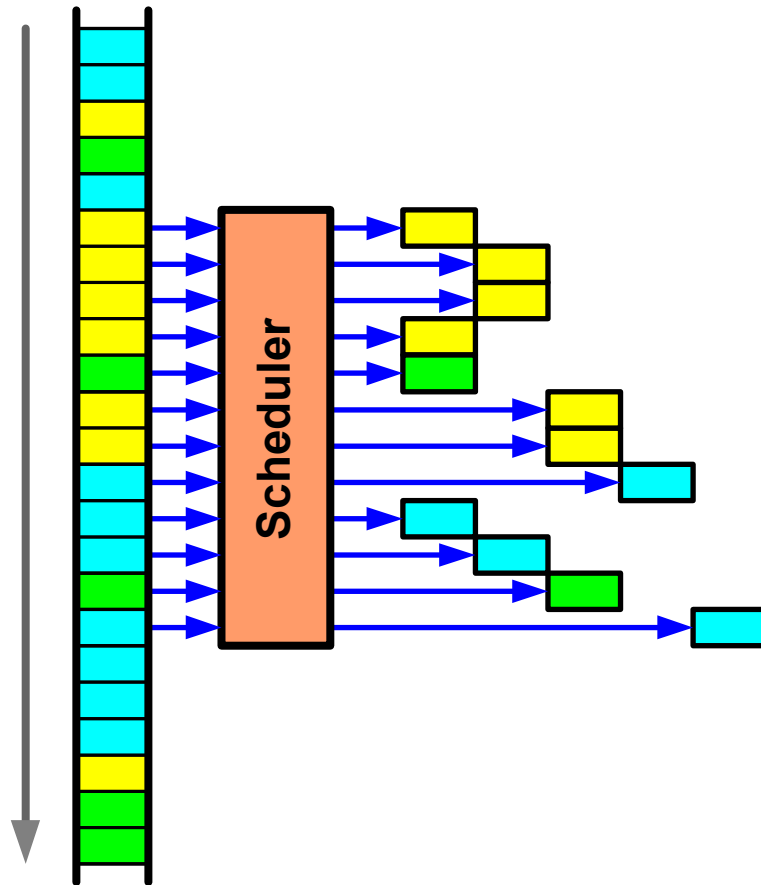
- Explizite Parallelität auf Befehlsniveau
  - Intel: **EPIC** (Explicitly Parallel Instruction Computing)
  - Welt: **VLIW** (Very Large Instruction Word)
- Massive Ressourcen
  - Umfangreicher Registersatz
  - Zahlreiche Ausführungseinheiten und Speicherports
- Prädikation
- Spekulative Ausführung
- Skalierbarkeit

# Vereinfachtes Blockschaltbild



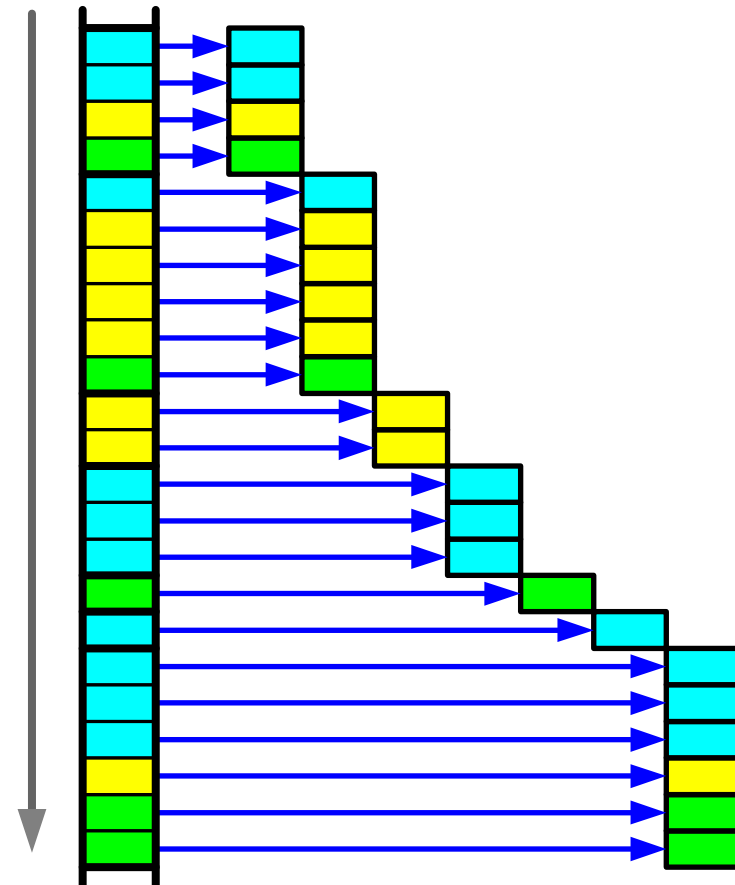
# Parallelität auf Befehlsniveau

Superskalar mit Out-Of-Order



Bernd Däne

EPIC bzw. VLIW



Was bringt der "Merced"?

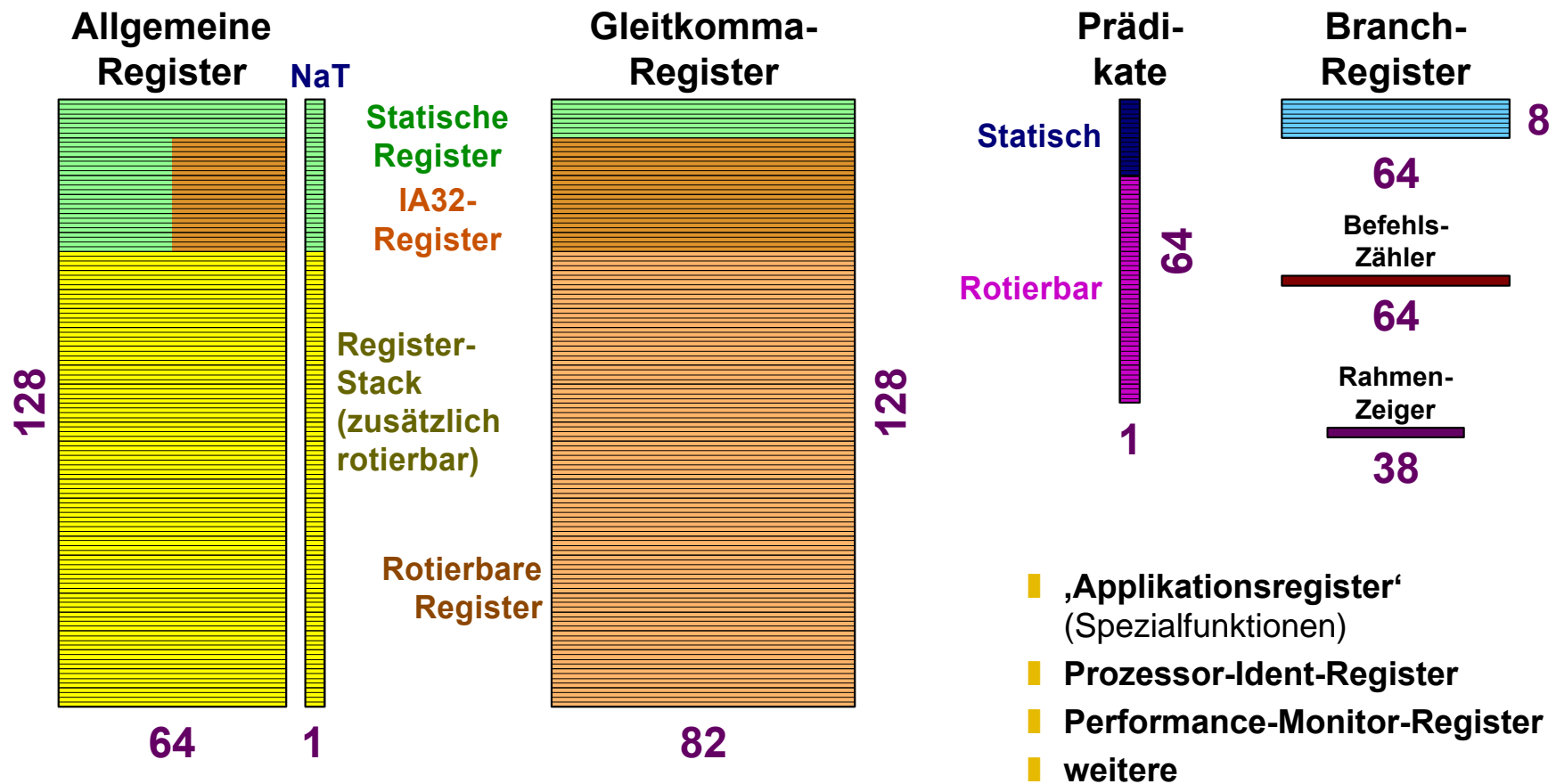
12

# Merkmale von EPIC bzw. VLIW

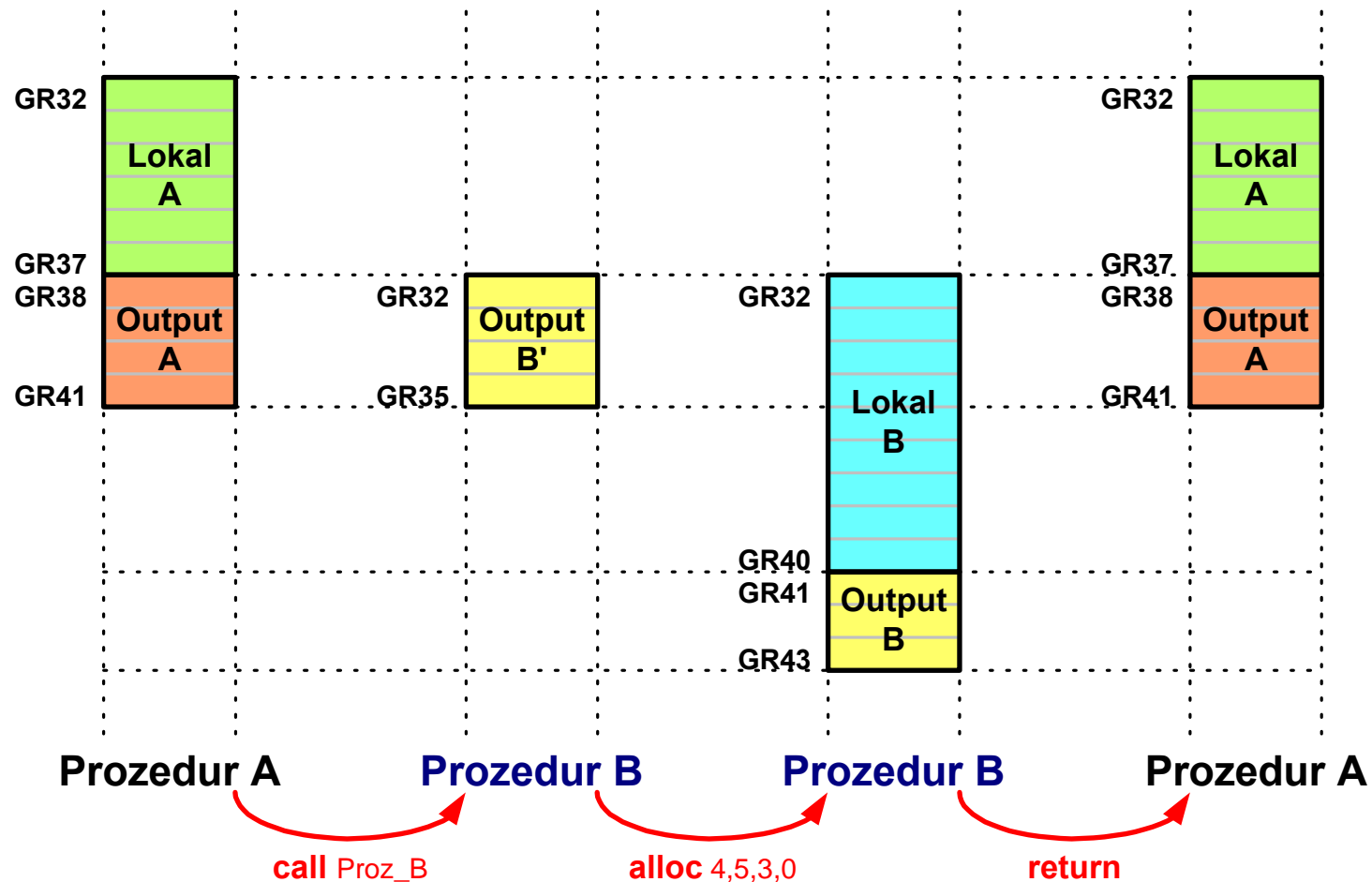
---

- Kein Struktur- und Zeitaufwand im Prozessor zur Extraktion von Parallelität
- Compiler sieht größeres Codefenster und kann effektiver optimieren
- Auflösung von Daten- und Ressourcenkonflikten bereits zur Übersetzungszeit
- Schlechtere Performance bei wechselnden Operations- und Speicherlatenzen
- Keine Kompatibilität zu früheren Befehlssätzen möglich

# Registersatz



# Arbeitsweise des Registerstacks

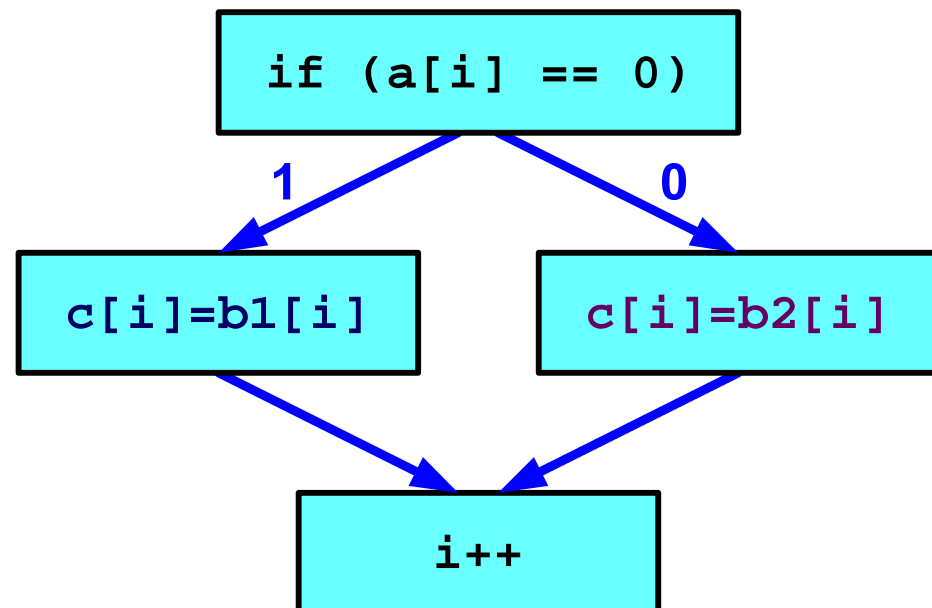


# Beispiel zur Prädikation (1)

## Quelltext

```
if (a[i] == 0)
    c[i] = b1[i];
else
    c[i] = b2[i];
i++;
```

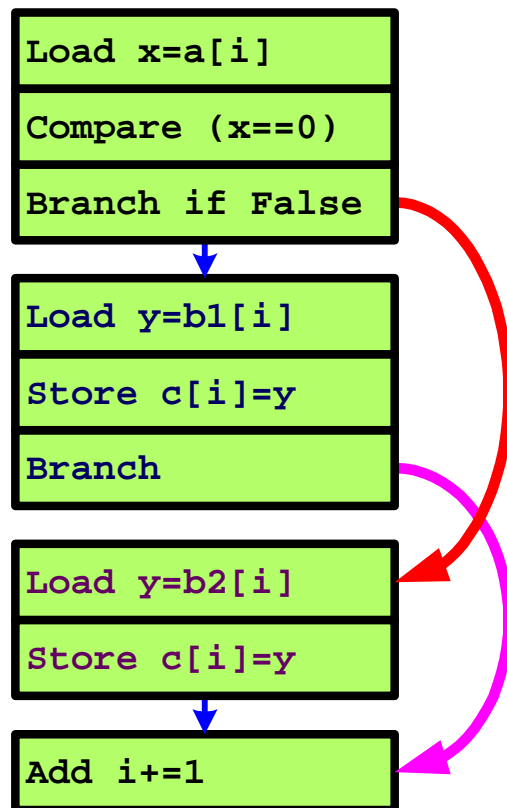
## Ablaufgraph



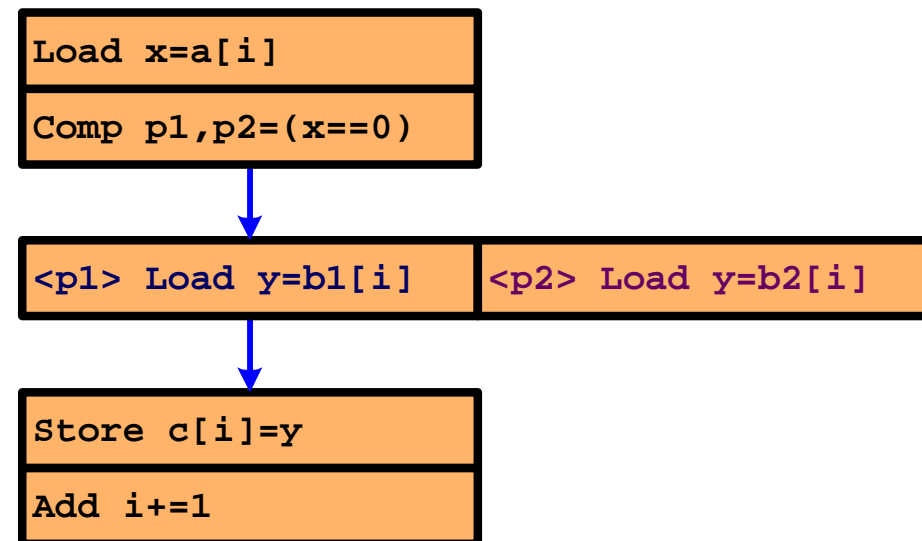


# Beispiel zur Prädikation (2)

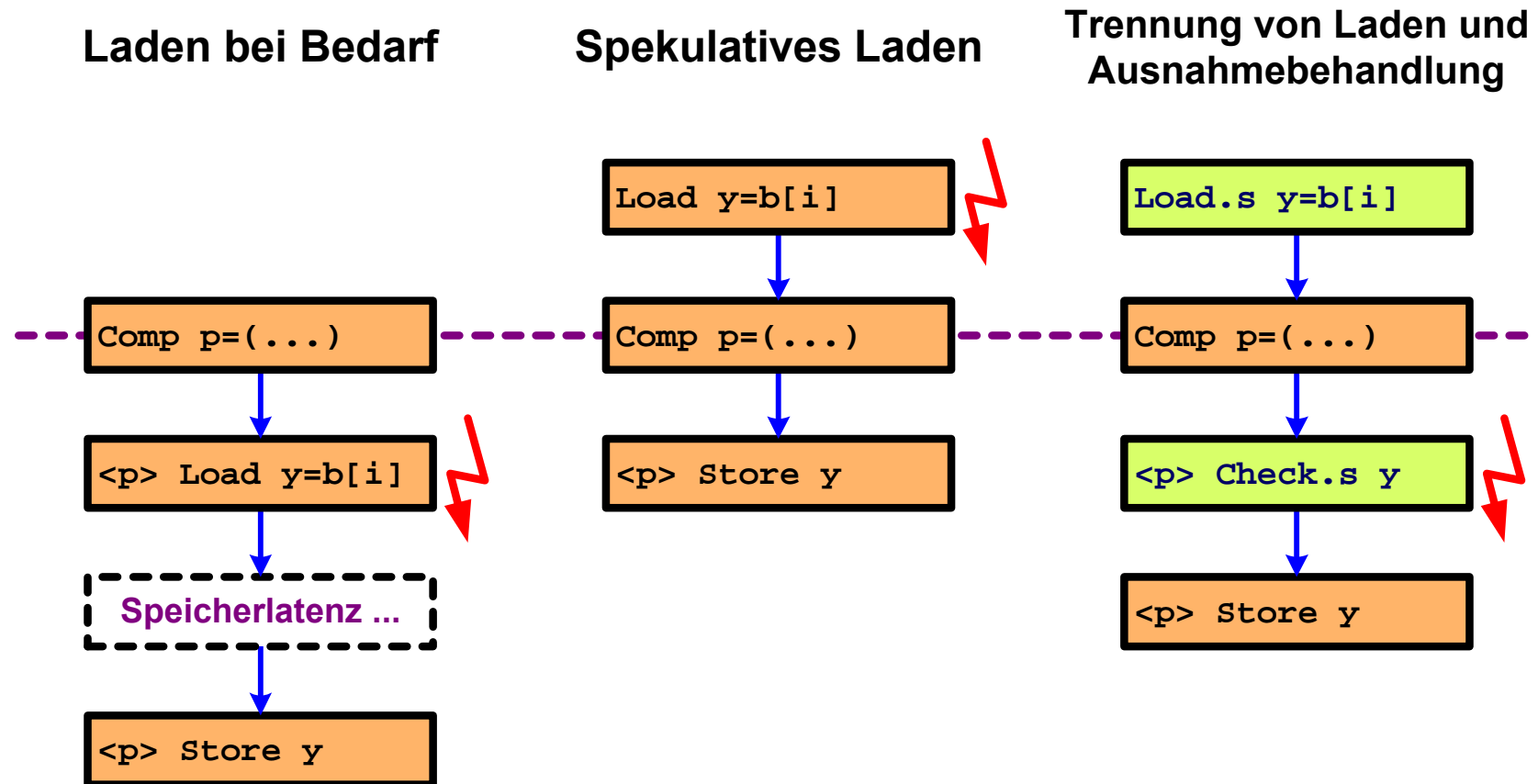
## Traditionell



## Prädikatiert

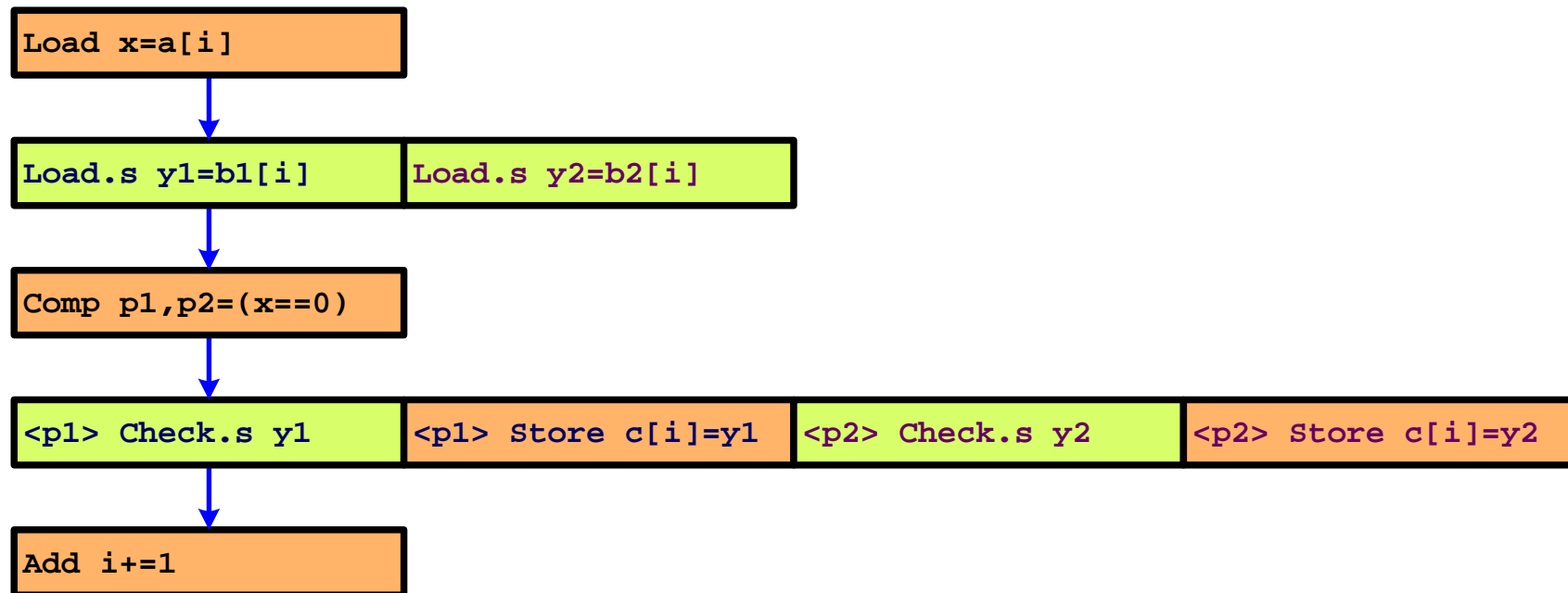


# Spekulativer Speicherzugriff (1)



# Spekulativer Speicherzugriff (2)

Resultierender Code für das Beispiel:



# Schleife entrollen: Beispiel

## Quelltext

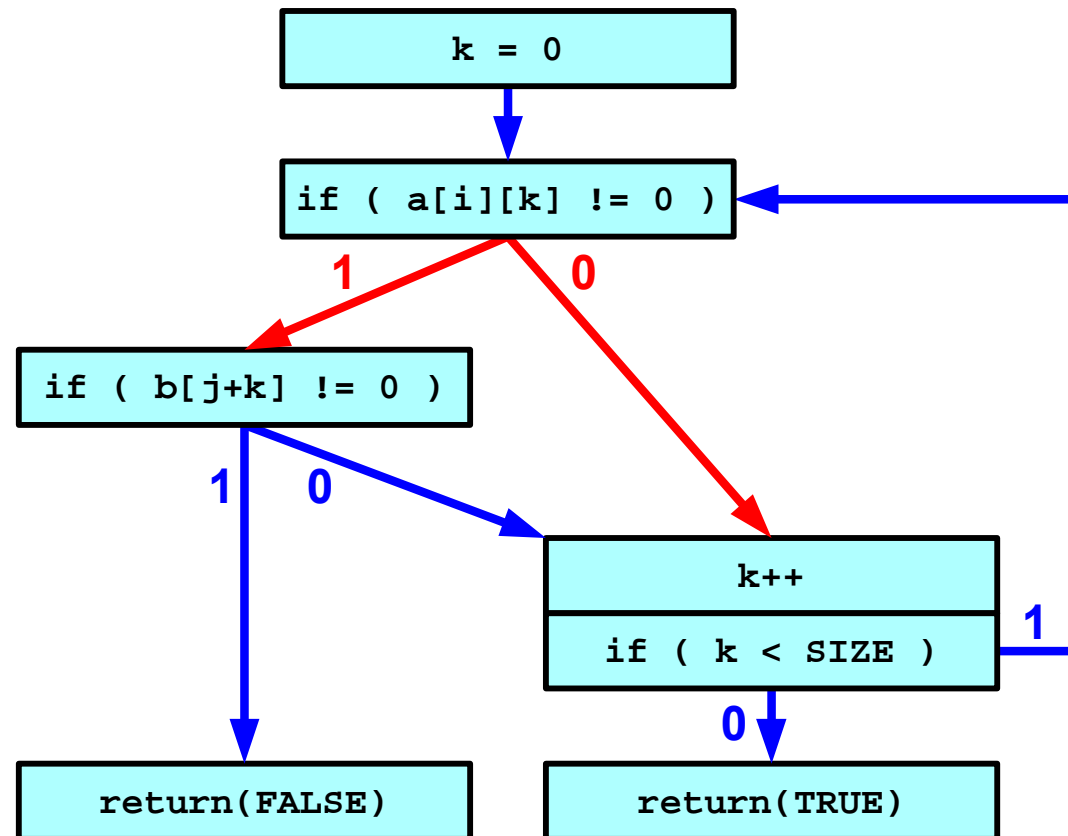
```
int a[S1][S2],b[S3];

int fit(i,j)
int i,j;
{
  int k;

  for (k=0;k<SIZE;k++)
    if (a[i][k])
      if (b[j+k])
        return (FALSE);

  return(TRUE);
}
```

## Ablaufgraph



# Schleife entrollen: Erstes „if“

Einheit Zyklus	I+M	I+M	I	I	B	B
prolog:	a_poi = &a[i][0];					
1	Load x = *a_poi		Add a_poi += 4			
2						
3	p1 = Comp (x != 0)					
4						
5						

# Schleife entrollen: Zweites „if“

Einheit Zyklus	I+M	I+M	I	I	B	B
<b>prolog:</b>	a_poi = &a[i][0]; b_poi = &b[j]; p2 = false;					
<b>1</b>	Load x = *a_poi	Load.s y = *b_poi	Add a_poi += 4	Add b_poi += 4		
<b>2</b>						
<b>3</b>	p1 = Comp (x != 0)					
<b>4</b>	<p1> p2 = Comp (y != 0)	<p1> Check.s y			<p2> Branch to ret_0	
<b>5</b>						
<b>ret_0:</b>	return(FALSE);					

# Schleife entrollen: Zähler

Einheit Zyklus	I+M	I+M	I	I	B	B
<b>prolog:</b>	a_poi = &a[i][0]; b_poi = &b[j]; p2 = false; k = 0;					
<b>1</b> <sup>L_1:</sup>	Load x = *a_poi	Load.s y = *b_poi	Add a_poi += 4	Add b_poi += 4		
<b>2</b>						
<b>3</b>	p1 = Comp (x != 0)	Add k += 1				
<b>4</b>	<p1> p2 = Comp (y != 0)	<p1> Check.s y	p3 = Comp (k >= SIZE)		<p2> Branch to ret_0	<p3> Branch to ret_1
<b>5</b>						Branch to L_1
<b>ret_1:</b>	return(TRUE);					
<b>ret_0:</b>	return(FALSE);					

# Schleife entrollen: Durchlauf Zwei

Einheit Zyklus	I+M	I+M	I	I	B	B
<b>prolog:</b>	a_poi = &a[i][0]; b_poi = &b[j]; p2 = false; k = 0;					
<b>1</b> <sup>L_1:</sup>	Load x = *a_poi	Load.s y = *b_poi	Add a_poi += 4	Add b_poi += 4		
<b>2</b>	Load.s x2 = *a_poi		Add a_poi += 4			
<b>3</b>	p1 = Comp (x != 0)	Add k += 1				
<b>4</b>	<p1> p2 = Comp (y != 0)	<p1> Check.s y	p3 = Comp (k >= SIZE)	p4 = Comp (x2 != 0)	<p2> Branch to ret_0	<p3> Branch to ret_1
<b>5</b>	Check.s x2					Branch to L_1
<b>ret_1:</b>	return(TRUE);					
<b>ret_0:</b>	return(FALSE);					



# Schleife entrollen: Fertig

Einheit Zyklus	I+M	I+M	I	I	B	B
<b>prolog:</b>	a_poi = &a[i][0]; b_poi = &b[j]; p2 = false; k = 0; p5 = false;					
<b>1</b> <sup>L_1:</sup>	Load x = *a_poi	Load.s y = *b_poi	Add a_poi += 4	Add b_poi += 4		
<b>2</b>	Load.s x2 = *a_poi	Load.s y2 = *b_poi	Add a_poi += 4	Add b_poi += 4		
<b>3</b>	p1 = Comp (x != 0)	Add k += 2				
<b>4</b>	<p1> p2 = Comp (y != 0)	<p1> Check.s y	p3 = Comp (k >= SIZE+1)	p4 = Comp (x2 != 0)	<p2> Branch to ret_0	<p3> Branch to ret_1
<b>5</b>	Check.s x2	<p4> Check.s y2	<p4> p5 = Comp (y2 != 0)	p6 = Comp (k < SIZE)	<p5> Branch to ret_0	<p6> Branch to L_1
<b>ret_1:</b>	return(TRUE);					
<b>ret_0:</b>	return(FALSE);					

# Das erste Foto

---

**Das Foto ist aus rechtlichen Gründen leider nicht enthalten.**

**Sie finden es im Internet:**

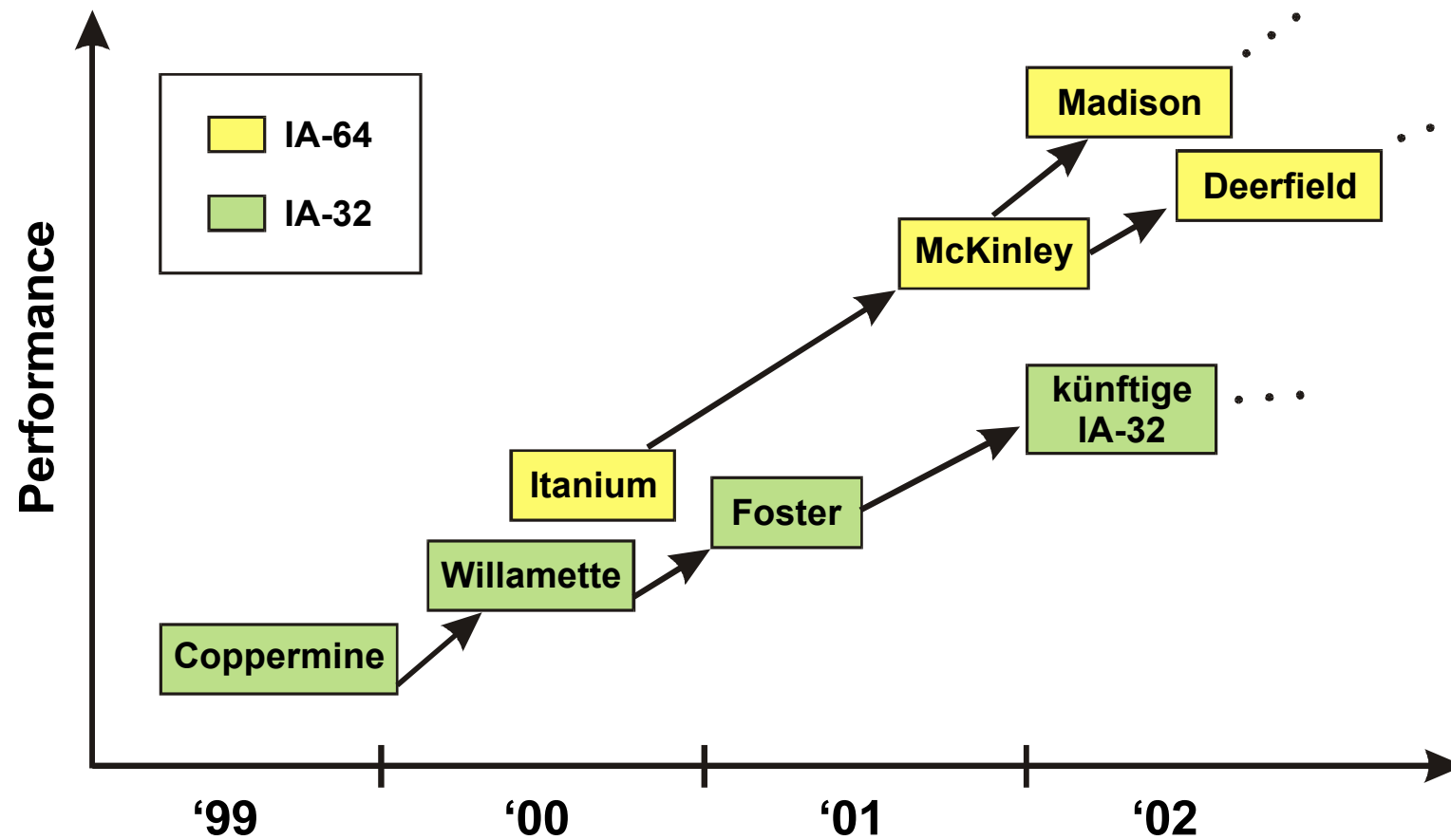
**<ftp://download.intel.com/pressroom/images/news/dp83199x.tif>**

# 5. Die Zukunft

---

- Markteinführung vermutlich Mitte 2000
- IA-64 zunächst nur für High-End
- IA-32 wird fortgesetzt (Intel und andere)
- Betriebssysteme für IA-64:
  - Win2000-64 (Microsoft)
  - Linux
  - Monterey (IBM), Solaris (SUN), ...
- Migration der Applikationen: langsam (?)

# Intel's Roadmap



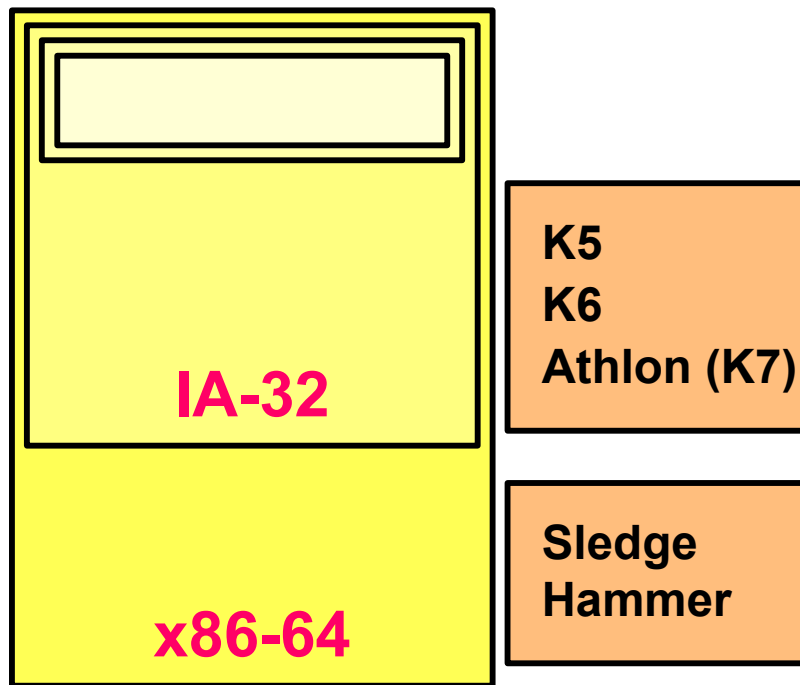
# 6. Die Anderen

---

- AMD: „**SledgeHammer**“ (Architektur: „x86-64“)
- Motorola: **G4+** (32 Bit), **G5** (32+64 Bit)
- IBM: **Power 4** (2 Cores/Chip, 4 Chips/Modul)
- Compaq/Digital: **Alpha EV8** (4-way SMT)
- SUN: **Sparc64 V** (superspekulativ)
- Transmeta: ??? (x86-zu-VLIW - Konverter?)

# AMD's Projekt „x86-64“

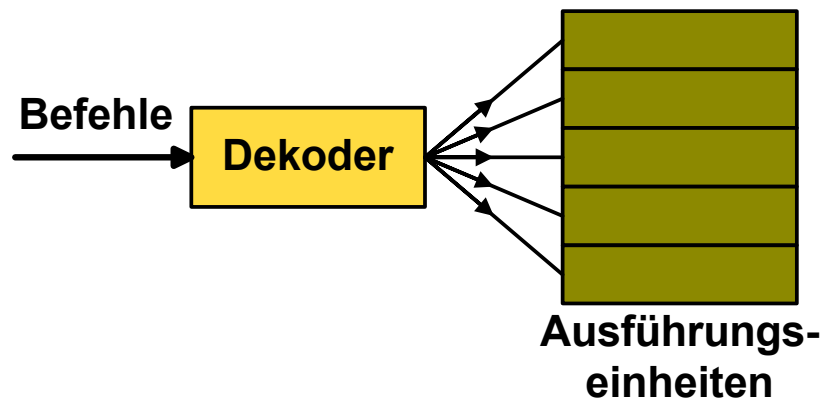
---



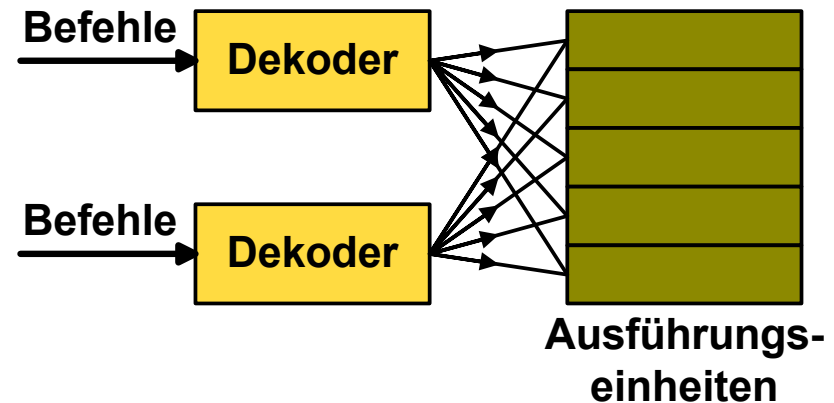
- Prinzip: direkte Erweiterung von IA-32
- Wenig Aufwand
- Volle Abwärtskompatibilität
- Nachteile von IA-32 bleiben erhalten
- Inkompatibel zu IA-64
- Bisher wenig Info

# Simultaneous Multithreading (SMT)

---



**Superskalar**



**SMT**

# Superspekulative Prozessoren

---

## ■ Spekulation zum Steuerfluss:

- Mehrstufige Sprungvorhersage (Trace-Analyse)

## ■ Spekulation zum Register-Datenfluss:

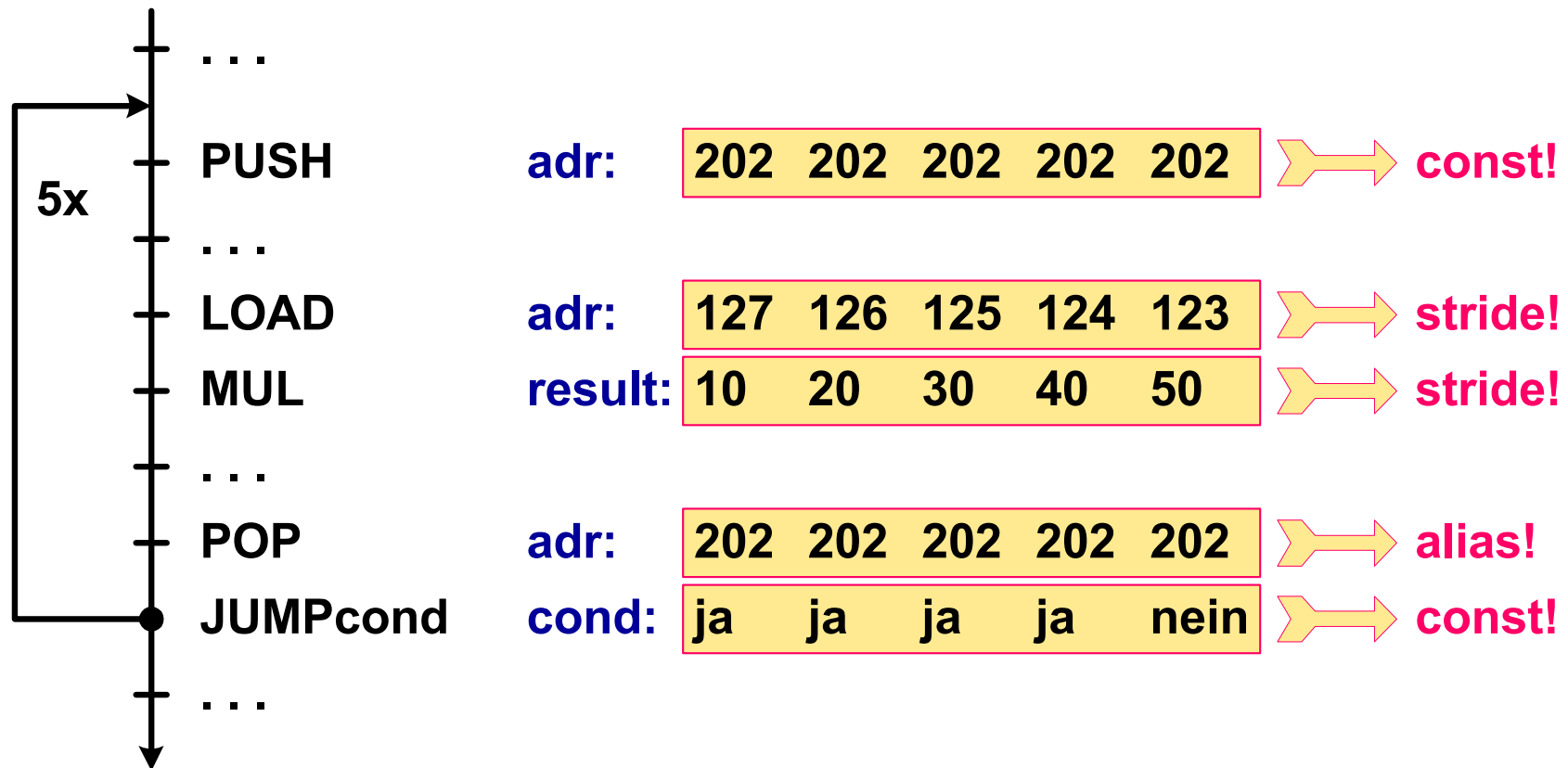
- Vorhersage von Datenunabhängigkeiten
- Vorhersage von Operationsergebnissen („value stride prediction“)

## ■ Spekulation zum Speicher-Datenfluss:

- Vorhersage von Lade-Adressen
- Vorhersage von Alias-Zugriffen



# Beispiel zur Superspekulation



# 7. Literatur und Web

---

- <http://developer.intel.com/design/ia64/>
- <ftp://download.intel.com/design/ia64/>
- Intels Weg zu 64 Bit. In: c't 12/99, S. 28  
<http://www.heise.de/ct/99/12/028/>
- Titanomachie. In: c't 22/99, S. 16  
<http://www.heise.de/ct/99/22/016/>
- The IA-64 Architecture at Work. In: IEEE Computer 7/98, S. 24
- <http://www.amd.com/news/prodpr/99105.html>
- Billion-Transistor Architectures. In: IEEE Computer 9/97, S. 46
  
- Dieser Foliensatz:  
<http://www.theoinf.tu-ilmenau.de/ra1/ver/>